

# Java Programming and Design

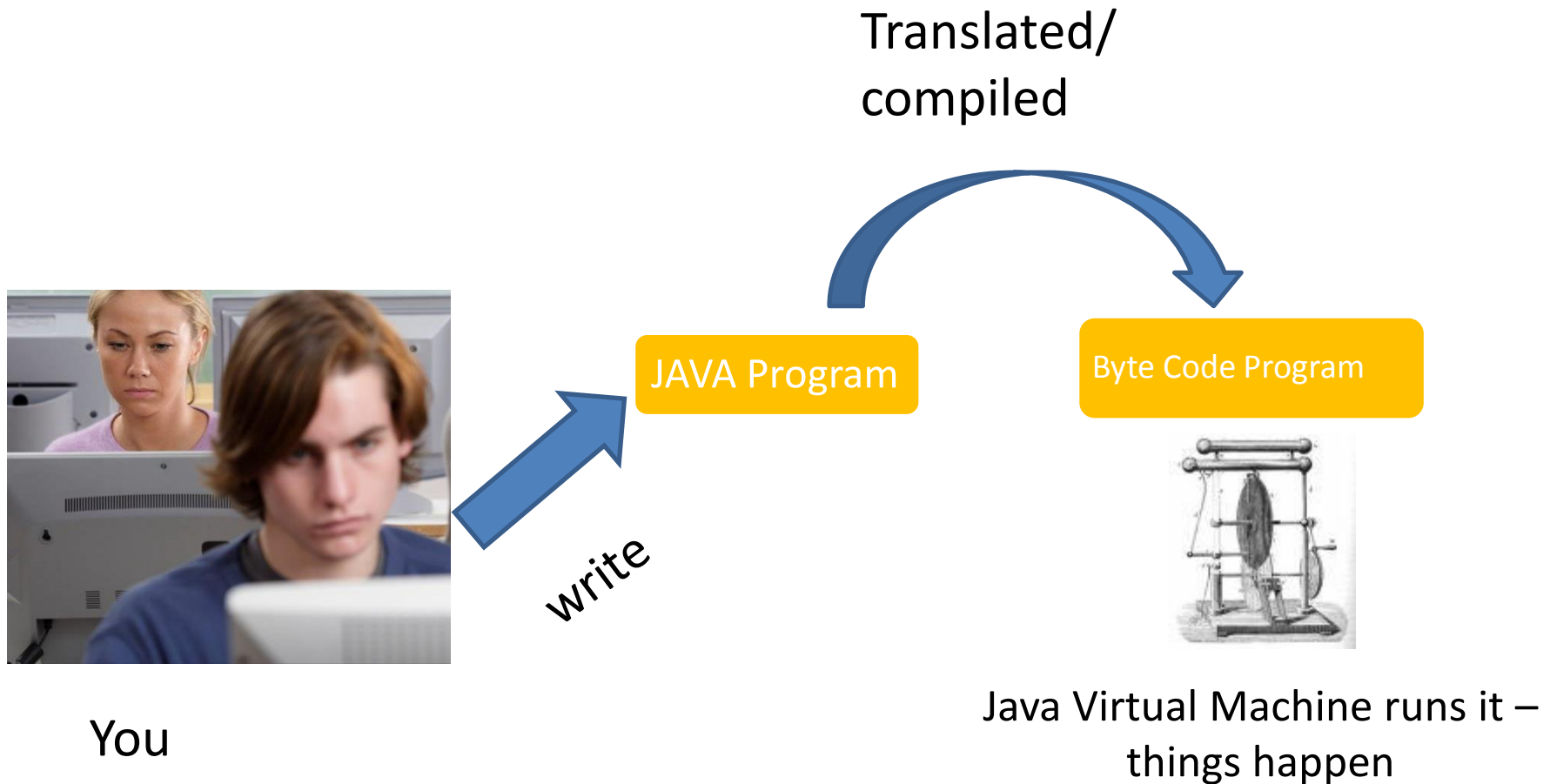
## Lecture 2 – Review, IntelliJ and Parameters

Chris Loftus, B62  
cwl@aber.ac.uk

# Qwizdom in lectures

- We only have a 100 seat license so please pair up an one person use smartphone etc
  - [qvr.qwizdon.com](http://qvr.qwizdon.com) Q5VN94
- When question asked discuss as a pair and then select an answer
- We will then discuss

# Remember our Overview of Java



Quiz: Why do we often use a compiler?  
(Choose all that apply)



- A) To check for errors in our code before we run it
- B) To run our code
- C) To translate our code to a more efficient form
- D) Not sure

# Quiz: Why is the Java Virtual Machine Virtual?



(Choose most accurate answer)

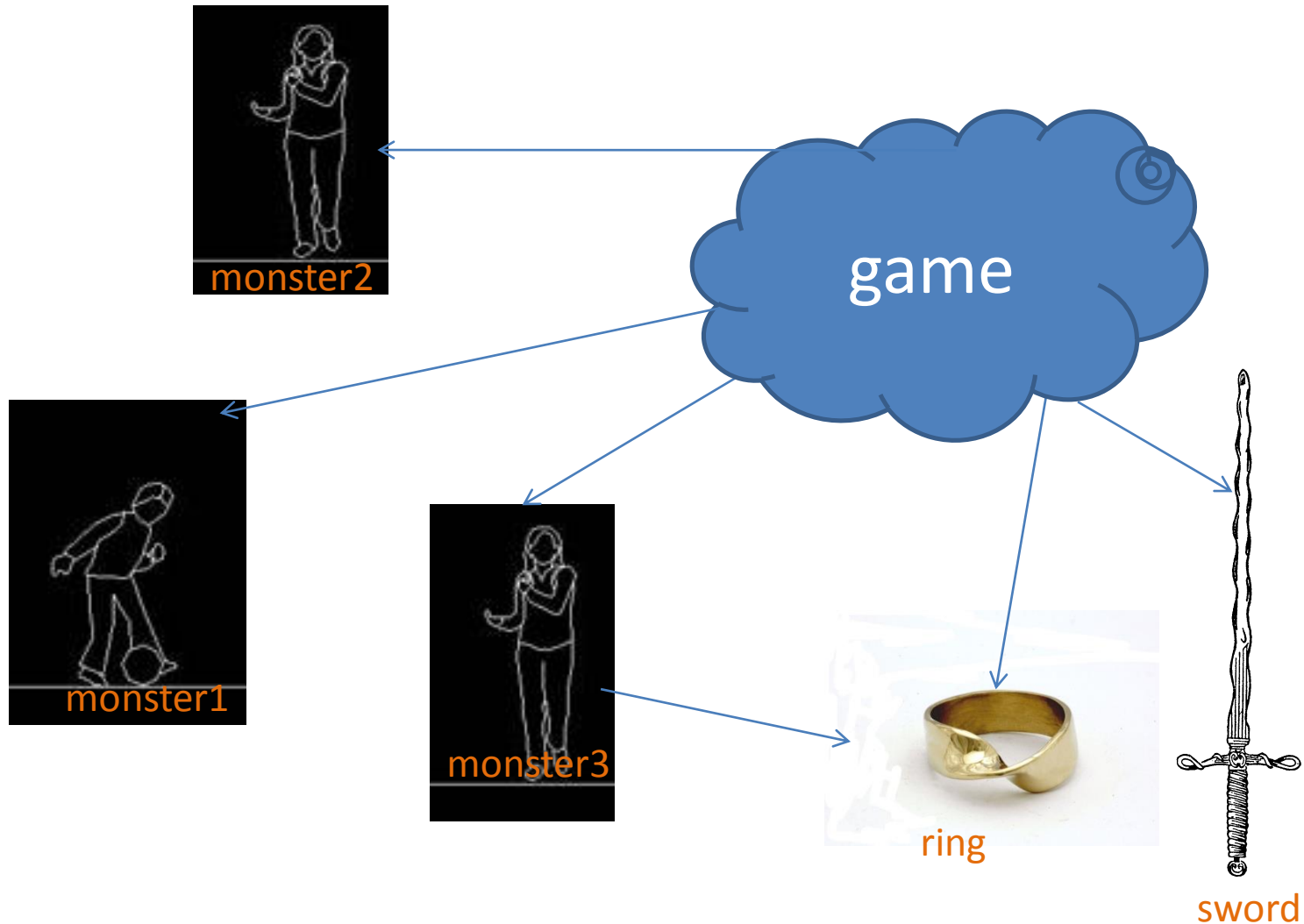
- A) It simulates a real computer so that our byte-code runs everywhere
- B) Because it doesn't really exist
- C) Not sure

Quiz: How is an interpreted language different from a compiled language?  
(Choose all that apply)

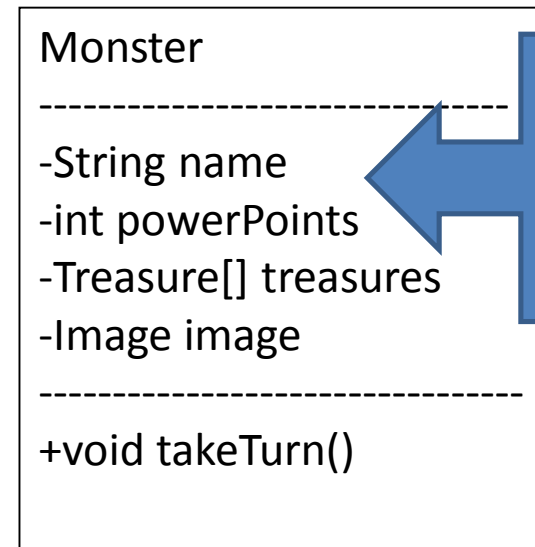
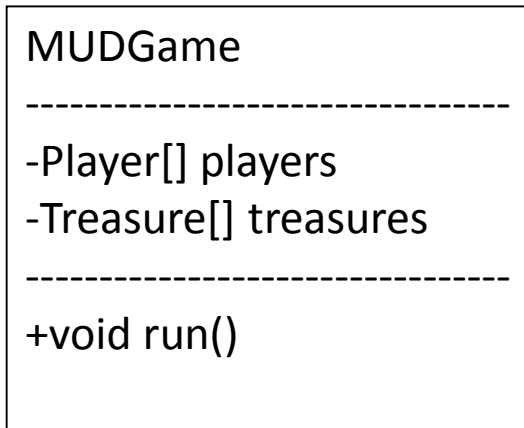


- A) It is usually more efficient and runs faster
- B) It is usually less efficient and runs slower
- C) It allows for more errors to occur at runtime
- D) Interpreted programs are quicker to change ready for running
- E) Not sure

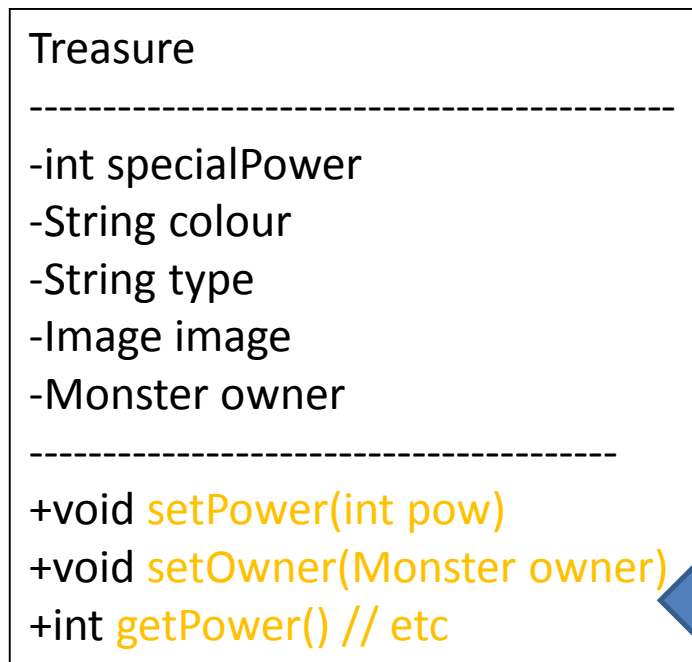
# Our object diagrams (as the program runs)



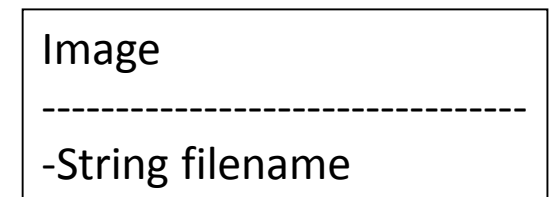
# and our Class diagram (for design)



These are  
the  
instance  
variable  
definitions



These  
are the  
methods





# So what did IntelliJ give us?

The screenshot displays the IntelliJ IDEA IDE interface. The top toolbar includes icons for saving, running, and debugging. The left sidebar shows the 'Project' window with the 'monster' project structure, including 'src' and 'out' folders. The main editor area shows the 'Application.java' file with the following code:

```
1  /**
2   * Created by chrisloftus on 05/01/2017.
3   */
4  public class Application {
5      public static void main(String[] args){
6          // Our starting code will go here
7          Monster monster1 = new Monster("elf", "green", 12);
8          Monster monster2 = new Monster("witch", "white", 10);
9          monster1.wail();
10         System.out.println(monster1.toString());
11     }
12 }
13
```

The bottom panel shows the 'Run' console output:

```
Run Application
/Library/Java/JavaVirtualMachines/jdk1.8.0_31.jdk/Contents/Home/bin/java
I am a elf watch me WAIL!!!
ooooooooo000000h
ooooooooo000000h
This scary monster is of type elf with green hair and 12 points
```

Three blue callout boxes highlight key features:

- Project Window**: Points to the left sidebar showing the project structure.
- The code editor**: Points to the main editor area displaying the Java code.
- Output Run console**: Points to the bottom panel showing the execution output.

# Also visual debugging

The screenshot shows the Eclipse IDE interface. The top editor displays the `Application.java` file with the following code:

```
public class Application {  
    /**  
     * This is the starting place for your program.  
     * The JVM will call this method when you run the program from  
     * Eclipse or from the command line: java Application  
     * We will say more about this method during a later talk,  
     * e.g. what static means.  
     * @param args Is an array that contains all the command line args;  
     * arguments, e.g. java Application arg1 arg2 arg3  
     */  
    public static void main(String[] args) { args: {}  
        // ADD loopy-circles.txt CODE HERE  
        Circle circle1 = new Circle(); circle1: Circle@441  
        Canvas c = Canvas.getCanvas();  
  
        int width = c.getWidth();  
        int height = c.getHeight();  
        int xPos = 0;  
        int currentXPos = 0;  
        int yPos = 0;  
    }  
}
```

A breakpoint is set on line 15, which is highlighted in blue. The left sidebar shows the project structure with `src` containing `Application`, `Canvas`, `Circle`, `Square`, and `Triangle`.

The bottom panel shows the **Debug** view with the **Debugger** tab active. The **Frames** list shows `main:16, Application`. The **Variables** list shows:

- `args = {String[0]@438}`
- `circle1 = {Circle@441}`
  - `diameter = 30`
  - `xPosition = 20`
  - `yPosition = 60`
  - `color = "blue"`
  - `isVisible = false`

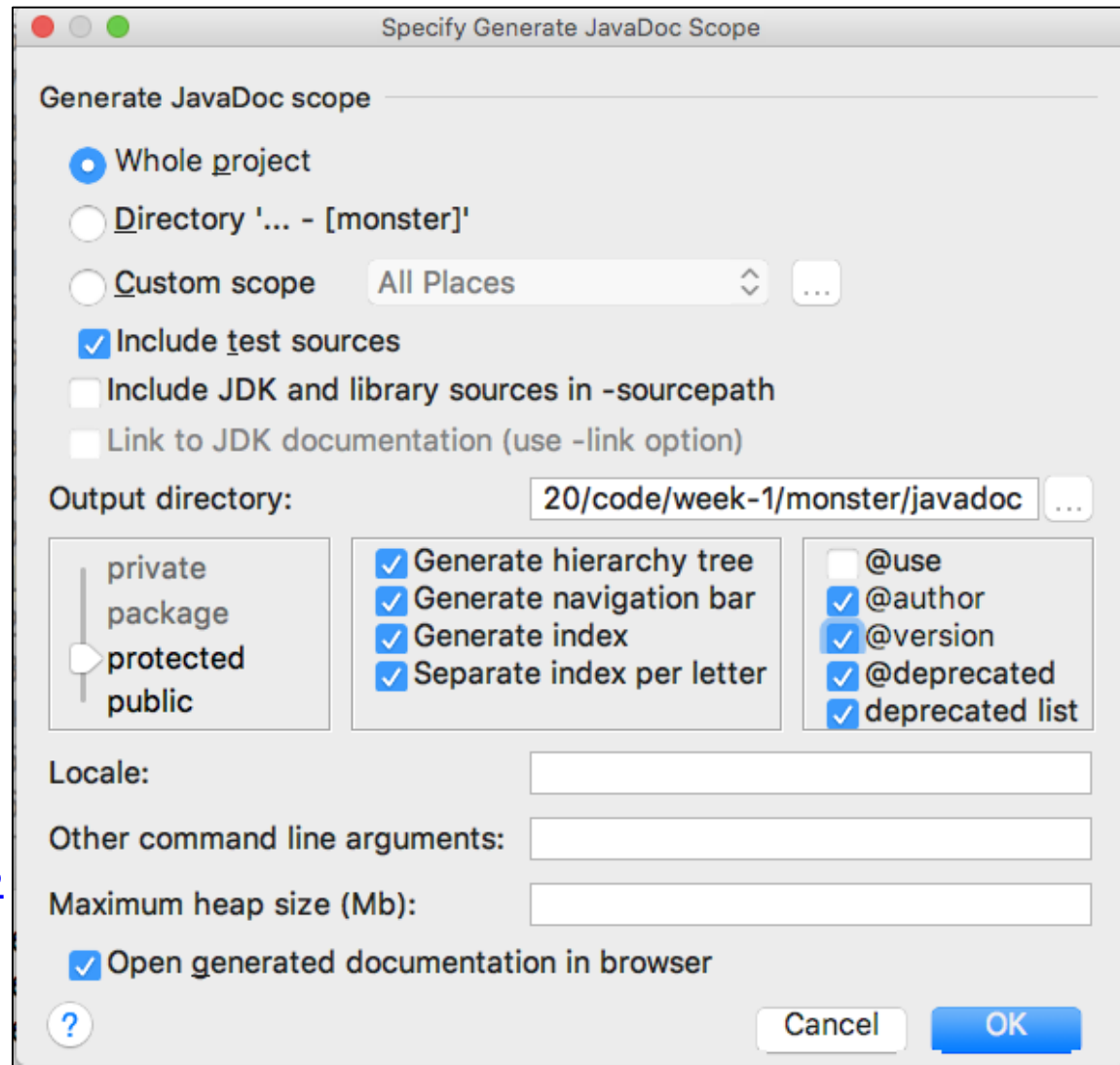
Three blue callout boxes provide instructions:

- You can set breakpoints** (points to the breakpoint icon on line 15)
- And step through your code (step-over highlighted)** (points to the step-over icon in the debugger toolbar)
- I expect you to use the visual debugger** (points to the entire debugger panel)

# And generating Java API documentation

- Generated from JavaDoc comments (next slide)
- Tools->Generate JavaDoc
- Same style as official Oracle JavaDoc API

<http://docs.oracle.com/javase/8/docs/api/>



```
/**
 * This is the class Monster – a 'blueprint' for all Monsters
 *
 * @author Chris Loftus
 * @version 1, 6th January 2017
 */
public class Monster {

    /**
     * Constructor for objects of class Monster
     */
    public Monster() {...}

    /**
     * Constructor for objects of class Monster
     *
     * @param startType    becomes initial type of monster
     * @param startHair    becomes initial hair colour
     * @param startPoints  initial power points
     */
    public Monster(String startType, String startHair, int startPoints) {...}

    /**
     * This method 'returns' information about a monster
     *
     * @return monster info for printing
     */
    public String toString() {...}

    // Other code omitted
}
```

JavaDoc  
comments start  
with `/**`

Always provide a  
header JavaDoc  
comment

Every non-private  
constructor / method  
should have one

`@param` for a  
parameter and  
`@return` for a return  
value

## Class Monster

java.lang.Object  
Monster

```
public class Monster  
extends java.lang.Object
```

This is the class Monster - a 'blueprint' for all Monsters

Version:

1, 6th January 2017

Author:

Chris Loftus

### Constructor Summary

#### Constructors

##### Constructor and Description

**Monster()**

Constructor for objects of class Monster

**Monster**(java.lang.String startType, java.lang.String startHair, int startPoints)

Constructor for objects of class Monster

### Method Summary

#### All Methods

#### Instance Methods

#### Concrete Methods

##### Modifier and Type

##### Method and Description

java.lang.String

**getHair()**

This method gets the hair colour

## Monster

```
public Monster(java.lang.String startType,  
               java.lang.String startHair,  
               int startPoints)
```

Constructor for objects of class Monster

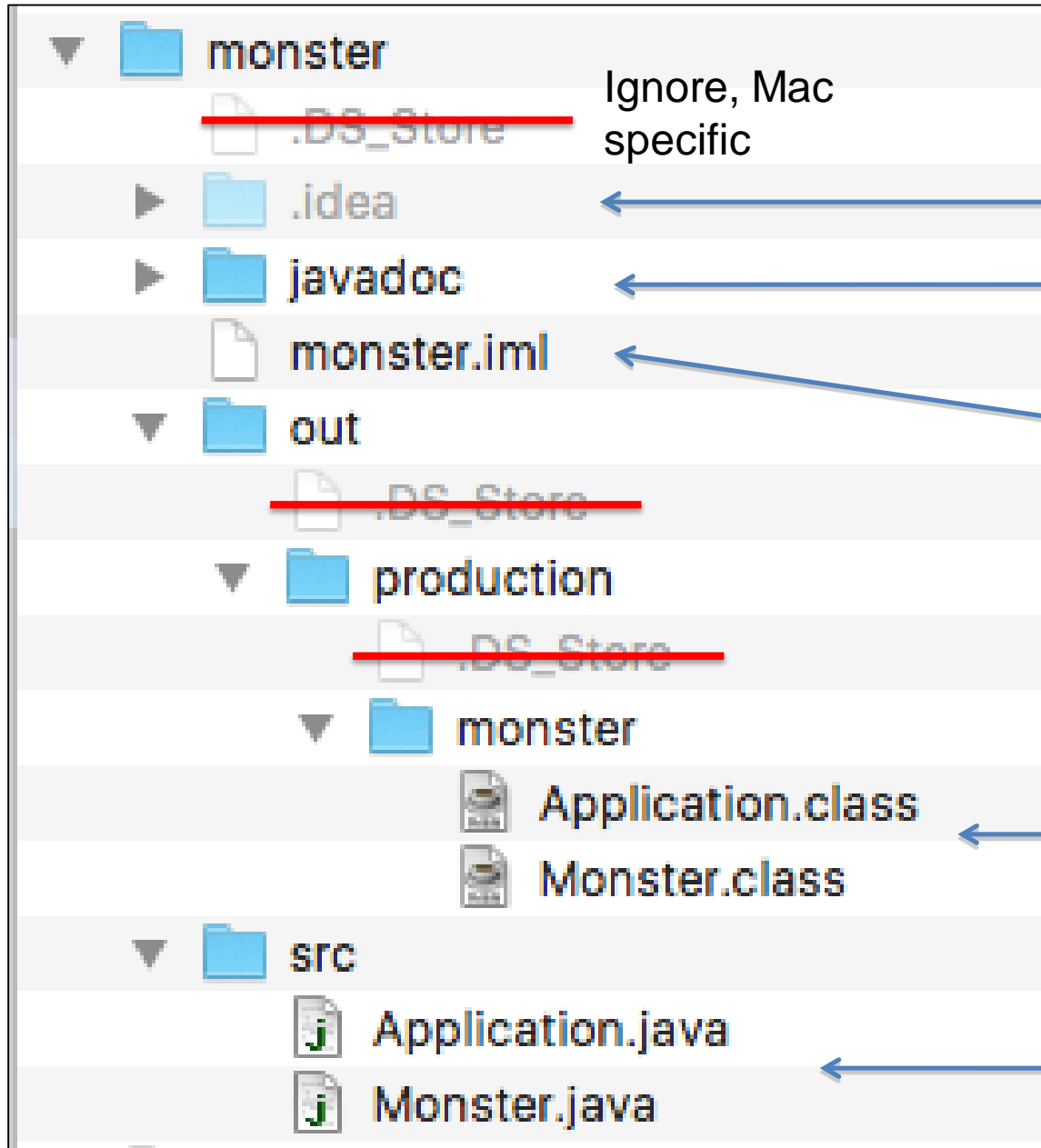
Parameters:

`startType` - becomes initial type of monster

`startHair` - becomes initial hair colour

`startPoints` - initial power points

# What files did IntelliJ give you?



# Things I didn't tell you

- Comments:
  - everything between `/* ..... */`
  - everything between `/** ..... */`
  - or rest of line from `//`
- Variable/method names mustn't have spaces and should start with lower case e.g. `specialPower` (uppercase for other words)
- Class names should start with upper case e.g. `HelloWorld`
- Return from methods...
  - `public String toString()` vs. `public void setName(String n)`



# What “methods” should you always have?

- Constructor `public Person() {...`
- `toString()` `public String toString() {...`
- Gets `public String getName() {...`
- Sets `public void setName (String n) {...`

There are a couple more (e.g. equals) that we'll come back to...

But in order to do anything ‘interesting’ we need more

# More on parameters

How you get a value into a method/constructor to change instance variables?

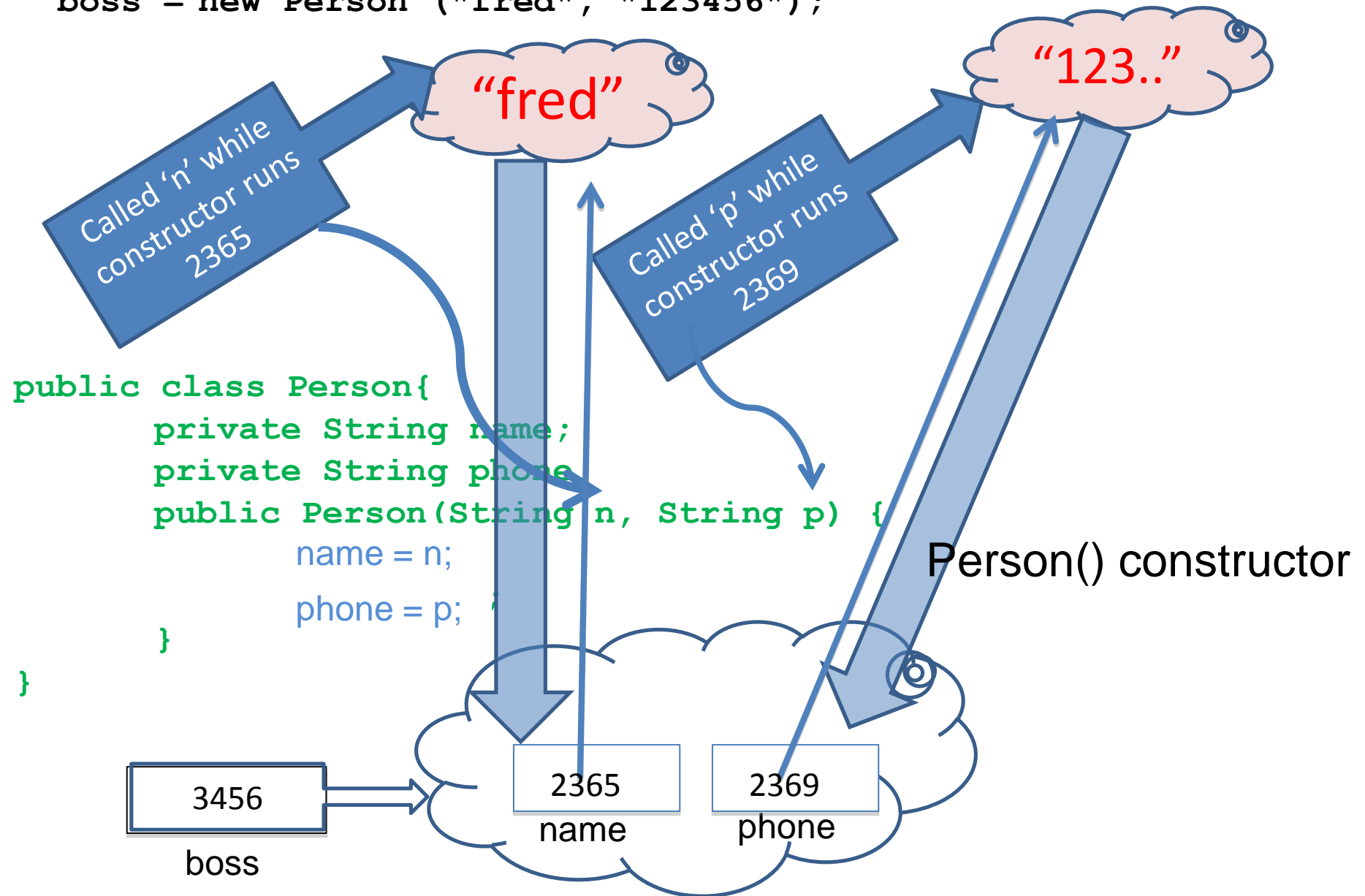
```
public class Person{  
    private String name;  
    private String phone;  
    public Person(String n, String p) {  
        name = n;  
        phone = p;  
    }  
}
```

Call this as:

```
Person boss = new Person("fred", "123456");
```

What happens?

```
Person boss;  
boss = new Person ("fred", "123456");
```



# We have done a lot!

- What is a class?
- What is an object?
- Design a class
- Looked at the code for it
- Learned to use IntelliJ basics

# Workshops this week

## Workshop 3

- Some more qwizdom
- **Small, assessed in-class exercise** on finding classes and links. Similar to last Monday's lecture in-class exercise: signed off by demonstrator and uploaded to BB
- Worksheet 7: Continue

## Workshop 4

- Talk on relationships and UML class diagrams
- **Final opportunity to sign-off worksheets 6 + 7**
- Worksheet 8